(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 March 2002 (07.03.2002)

PCT

(10) International Publication Number
WO 02/19592 A2

(51) International Patent Classification[7]: H04L

(21) International Application Number: PCT/KR01/00259

(22) International Filing Date: 21 February 2001 (21.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
2000/49884 26 August 2000 (26.08.2000) KR

(71) Applicants and
(72) Inventors: **PARK, Tae-Kyou** [KR/KR]; 130-303 Hanvit Apartment, Eoeun-dong, Yuseong-gu, Daejeon 305-755 (KR). **IM, Yeon-Ho** [KR/KR]; 102-304 Dure Apartment, Sinseong-dong, Yuseong-gu, Daejeon 305-720 (KR). **JO, In-Gu** [KR/KR]; 1-101, 402-11(30/4), Galma-dong, Seo-gu, Daejeon 302-170 (KR).

(74) Agents: **CHO, Jin-Su** et al.; Byukcheon B/D. 4F, 1597-5, Seocho-dong, Seocho-gu, Seoul 137-876 (KR).

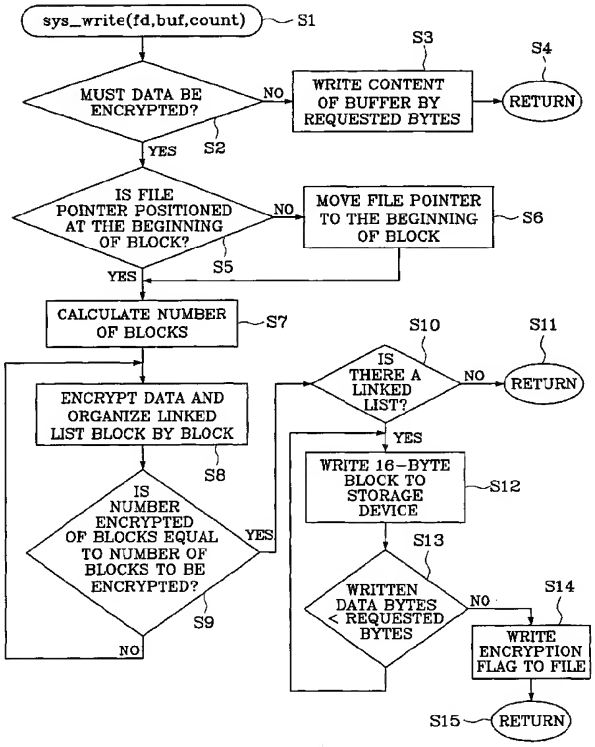
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: METHOD OF AUTOMATICALLY ENCRYPTING AND DECRYPTING FILE IN KERNEL MODE, METHOD OF MOVING FILE POINTER USING THE SAME, AND COMPUTER READABLE RECORDING MEDIUM ON WHICH PROGRAMS OF ABOVE METHODS ARE RECORDED



(57) Abstract: A method of automatically encrypting and decrypting a file in kernel, a method of moving a file pointer using the same, and a computer readable recording medium on which programs produced by programming the above methods are recorded are disclosed. In the method of automatically encrypting and decrypting a file in kernel, when a user process of a UNIX operating system intends to write a file to a storage device such as a disk, diskette, or CD-ROM, the file is automatically encrypted block by block, and stored in the storage device in kernel mode of the operating system without respect to user's intention, and when the user process intends to read the encrypted file, the whole data of the file or a necessary portion of the data is automatically decrypted block by block in kernel mode. In addition, when a file pointer is moved in an encrypted file according to the method of moving a file pointer, the amount of file movements is calculated while the file is automatically decrypted block by block according to the method, and then the file pointer is moved based on the calculated value. The recording medium is a predetermined recording medium on which programs produced by programming the method of automatically encrypting and decrypting a file in kernel, and the method of moving a file pointer using the same are recorded.

WO 02/19592 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD OF AUTOMATICALLY ENCRYPTING AND DECRYPTING FILE IN KERNEL MODE, METHOD OF MOVING FILE POINTER USING THE SAME, AND COMPUTER READABLE RECORDING MEDIUM ON WHICH PROGRAMS OF ABOVE METHODS ARE RECORDED

TECHNICAL FIELD

The present invention relates to a file security method for a UNIX operating system, and more particularly, to a method of automatically encrypting and storing data in kernel mode when a user process intends to write data to a memory device, and automatically decrypting the whole or a required portion of the encrypted data in kernel mode when a user process intends to read the encrypted data, a method of moving a file pointer using the same, and a computer readable recording medium on which programs produced by programming the above methods are recorded.

BACKGROUNG ART

Recently, since secret information or commercially valuable information stored in a storage device, for example, an auxiliary storage device of a computer may be illegally copied or stolen via a computer network or a storage medium, or may be lost or leak out due to mistakes of a owner of the information, security problems become more serious and important.

In a conventional encryption and decryption in an operating system, a user encrypts and decrypts a file as a whole with an encryption and decryption key by using an encryption program after the user selects the file.

However, there are the following problems in such a method.

First, when a secret file is produced, a user may not encrypt the file erroneously or intentionally.

Second, when a file is read or modified in a document producing program, an accounting program or another application program, since the whole file must be decrypted and the decrypted file must be saved for further processing, any secret

file may be left saved in an auxiliary storage device in a decrypted form.

Third, in the aspect of a processing speed, the encryption and decryption processes in user mode take more time than those in kernel mode.

Fourth, a separate key managing method must be provided for managing all the encryption and decryption keys.

Consequently, the encryption and decryption operations in user mode make a user cumbersome, and include a problem such as weakness in security, that is, possibility of leakage of a secret file.

DISCLOSURE OF THE INVENTION

To solve the above problems, it is an objective of the present invention to provide a method of automatically encrypting and storing a file in kernel mode of a UNIX operating system when a user process intends to write a file, and automatically decrypting the whole file or a required portion of the encrypted data in kernel mode when a user process intends to read the encrypted file.

It is another objective of the present invention to provide a method capable of automatically and compulsively encrypting and decrypting a security-classified file with a key of a system security manager embedded in the kernel without omitting to encrypt a security-classified file due to a user's error and fundamentally without respect to user's encryption designation.

It is still another objective of the present invention to provide an improved encryption and decryption method which is capable of encrypting and decrypting a portion or portions of a file required to be encrypted in a document producing program, an accounting program, or another application program, and need not decrypt a whole file.

It is still another objective of the present invention to provide a method of encrypting and decrypting a file that can be performed at a higher speed by not using an encryption and

decryption key designated by a user but using a key embedded in a operating system when a file is encrypted or decrypted.

It is still another objective of the present invention to provide an improved method of invoking a file pointer moving system call using a method of decrypting a file according to the present invention.

It is still another objective of the present invention to provide a computer readable recording medium on which programs produced by programming methods according to the present invention are recorded.

Accordingly, In accordance with an embodiment of the present invention, there is provided a method of automatically encrypting, in kernel mode of a UNIX operating system, wherein a second number of bytes of data requested by a user process to be newly written by the block unit having a first number of bytes and including a first area in which data is actually written, and a second area in which the number of bytes of the data written in the first area is recorded, comprising the steps of: (a) calculating the number of blocks required for encrypting the second number of bytes of data while taking into consideration the number of bytes constituting the first area and the second number of bytes of data; (b) respectively for the each block, copying a proper portion of data requested to be written from a buffer transferred from the user process to the first area, writing the number of bytes of the data written in the first area to the second area, encrypting the data written in the first and second areas by the unit of the first number of bytes using a predetermined encryption algorithm, and organizing a linked list of corresponding block in memory with respect to the corresponding block; and (c) writing the linked list of the blocks organized in memory to a storage device.

In this embodiment, the encryption algorithm used in the step (b) is a block encryption algorithm.

Preferably, encryption key used in the block encryption algorithm is a key embedded in a kernel image when the kernel is compiled and generated.

Preferably, the encryption key is a seed encryption key having predetermined bits, which is produced by letting a user determine a series of unique letters for generating a key and then encrypting the series of unique letters by a predetermined encryption algorithm.

Preferably, whenever a write system call is invoked, the seed encryption key is re-encrypted using information, which is unique to a file and is included in an i-node of the file, and then the re-encrypted seed encryption key is used as an encryption key in the block encryption algorithm of the step (b).

In one version of this embodiment, before the step (a), the method further comprises the step of checking whether or not the present file pointer is positioned at the beginning of a block, and moving the file pointer to the beginning of the block depending on the result.

In another version of this embodiment, the second area is an area used to record the number of bytes of data written in the first area, and is allocated at a predetermined location in the block.

In another version of this embodiment, the number of blocks to be encrypted is calculated by dividing the second number of bytes of data by the number of bytes of the first area, and adding 1 to the integer quotient thereof.

In another version of this embodiment, after the step (c), the method further comprises the step of writing an encryption flag in a security level field allocated in advance in a reserved field of the i-node of an encrypted file to indicate that the file is encrypted.

In another version of this embodiment, before the step (a), the method further comprises the step of: (a1) letting a user input user information including security level information when the user logs in a UNIX system, and authenticating the user; (b1) writing the user's security level information in a security level field, having predetermined bits and allocated in a task structure of a user process when the user is authenticated in the step (a1); and (c1) determining whether or not the data

requested to be written is data which must be encrypted by confirming whether or not security level information is written in the security level field of the task structure. In this version of the present invention, after the step (c1), the method further may comprise the step of copying the user's security level information written in the task structure of the user process and writing the copied security level information in a reserved field allocated in a reserved field of the i-node of the file in which the data is encrypted and stored.

In accordance with another embodiment of the present invention, there is provided a method of automatically decrypting data block by block in kernel mode of a UNIX operating system when a system call to read a second number of bytes of data from a file which is encrypted by the block unit having a first number of bytes using a block encryption algorithm, and to store the data in a buffer is invoked, wherein the block including a first area in which encrypted data is written, and a second area in which the number of bytes of effective data written in the first area is encrypted and written, the method comprising the steps of: (a) adjusting the position of a file pointer to the beginning of a block by taking the time when the write system call is invoked as a reference point; (b) continuing to repeat a linked list organizing procedure until the total added number of bytes of effective data of the organized linked list is equal to the second number of bytes requested to read, wherein the linked list organizing procedure performed block by block comprising a copying operation in which a block is copied to memory; a decrypting operation in which the copied block is decrypted using the encryption key used when the file was encrypted; and a linked list organizing operation in which the linked list for the decrypted block is organized in memory with reference to the number of bytes of effective data of the decrypted block, written in the second area of the block ; and (c) copying the linked list obtained in the step (b) to the buffer.

In accordance with another embodiment of the present

invention, there is provided a method of moving a file pointer while automatically decrypting data block by block in kernel mode of a UNIX operating system when a system call to move the file pointer a second number of bytes in the positive direction from a predetermined byte location is invoked by a user process in a file in which data is encrypted by the block unit having a first number of bytes using a block encryption algorithm, wherein the block including a first area in which encrypted data is written, and a second area in which the number of bytes of effective data written in the first area is encrypted and written, the method comprising the steps of: (a) adjusting the present position of the file pointer to the predetermined byte location; (b) checking whether or not the position of the file pointer adjusted in the step (a) is the beginning of the corresponding block, and moving the file pointer to the beginning of the corresponding block according to the result; (c) continuing to repeat a linked list organizing procedure until the total added number of bytes of effective data of the organized linked list is equal to the second number of bytes requested from the user process, wherein the linked list organizing procedure performed block by block comprising a copying operation in which a block is copied to memory; a decrypting operation in which the copied block is decrypted using the encryption key used when the file was encrypted; and a linked list organizing operation in which the linked list for the decrypted block is organized in memory with reference to the number of bytes of effective data of the decrypted block, written in the second area of the block; and (d) calculating the actual number of bytes by which the file point must be moved actually by using the linked list obtained in the step (c), and actually moving the file pointer based on the calculated result.

In accordance with another embodiment of the present invention, there is provided a method of moving a file pointer while automatically decrypting data block by block in kernel mode of a UNIX operating system when a system call to move the file pointer a second number of bytes in the negative direction

from a predetermined byte location is invoked by a user process in a file in which data is encrypted by the block unit having a first number of bytes using a block encryption algorithm, wherein the block including a first area in which encrypted data is written, and a second area in which the number of bytes of effective data written in the first area is encrypted and written, the method comprising the steps of: (a) adjusting the present position of the file pointer to the predetermined byte location; (b) checking whether or not the position of the file pointer adjusted in the step (a) is the beginning of the corresponding block, and moving the file pointer to the beginning of the corresponding block according to the result; (c) continuing to repeat a linked list organizing procedure until the total added number of bytes of effective data of the organized linked list is equal to the second number of bytes requested by the user process, wherein the linked list organizing procedure performed block by block comprising a copying operation in which a block is copied to memory; a decrypting operation in which the copied block is decrypted using the encryption key used when the file was encrypted; a linked list organizing operation in which the linked list for the decrypted block is organized in memory with reference to the number of bytes of effective data of the decrypted block, written in the second area of the block; and a file pointer moving operation in which the file pointer is moved by double the size of the block in the negative direction; and (d) calculating the actual number of bytes by which the file point must be moved actually by using the linked list obtained in the step (c), and actually moving the file pointer based on the calculated result.

In accordance with another embodiment of the present invention, there is provided a computer readable recording medium on which a program is recorded, the program being capable of automatically encrypting, in kernel mode of a UNIX operating system, a second number of bytes of data requested by a user process to be newly written by the block unit having a first

number of bytes and including a first area in which data is actually written, and a second area in which the number of bytes of the data written in the first area is recorded, the program comprising: (a)a program module capable of calculating the number of blocks required for encrypting the second number of bytes of data while taking into consideration the number of bytes constituting the first area and the second number of bytes of data; (b) a program module capable of copying data requested to be written from a buffer transferred from the user process to the first area, and writing the number of bytes of the data written in the first area to the second area, encrypting, by the first number of bytes, the data written in the first and second areas using a block encryption algorithm, and organizing a linked list of corresponding block in memory, with respect to individual blocks, wherein the number of blocks for data to be written and encrypted is calculated by the program module (a); and (c) a program module capable of writing the linked list of the blocks organized in memory to a storage device.

In accordance with another embodiment of the present invention, there is provided a computer readable recording medium on which a program is recorded, the program being capable of automatically decrypting data block by block in kernel mode of a UNIX operating system when a system call to read a second number of bytes of data from a file which is encrypted by the block unit having a first number of bytes using a block encryption algorithm, and to store the data in a buffer is invoked, wherein the block including a first area in which encrypted data is written, and a second area in which the number of bytes of effective data written in the first area is encrypted and written, the program comprising: (a) a program module capable of adjusting the position of a file pointer to the beginning of a block by taking the time when the write system call is invoked as a reference point; (b) a program module capable of continuing to repeat a linked list organizing procedure until the total added number of bytes of effective data of the organized linked list is equal to the second number

of bytes requested to read, wherein the linked list organizing procedure performed block by block comprising a copying operation in which a block is copied to memory; a decrypting operation in which the copied block is decrypted using the encryption key used when the file was encrypted; and a linked list organizing operation in which the linked list for the decrypted block is organized in memory with reference to the number of bytes of effective data of the decrypted block, written in the second area of the block; and (c) a program module capable of copying the linked list obtained in the step (b) to the buffer.

In accordance with another embodiment of the present invention, there is provided a A computer readable recording medium on which a program is recorded, the program being capable of moving a file pointer while automatically decrypting data block by block in kernel mode of a UNIX operating system when a system call to move the file pointer a second number of bytes in the positive direction from a predetermined byte location is invoked by a user process in a file in which data is encrypted by the block unit having a first number of bytes using a block encryption algorithm, wherein the block including a first area in which encrypted data is written, and a second area in which the number of bytes of effective data written in the first area is encrypted and written, the program comprising: (a) a program module capable of adjusting the present position of the file pointer to the predetermined byte location; (b) a program module capable of checking whether or not the position of the file pointer adjusted in the step (a) is the beginning of the corresponding block, and moving the file pointer to the beginning of the corresponding block according to the result; (c) a program module capable of continuing to repeat a linked list organizing procedure until the total added number of bytes of effective data of the organized linked list is equal to the second number of bytes requested by the user process, wherein the linked list organizing procedure performed block by block comprising a copying operation in which a block is copied to

memory; a decrypting operation in which the copied block is decrypted using the encryption key used when the file was encrypted; and a linked list organizing operation in which the linked list for the decrypted block is organized in memory with reference to the number of bytes of effective data of the decrypted block, written in the second area of the block; and (d) a program module capable of calculating the actual number of bytes by which the file point must be moved actually by using the linked list obtained in the step (c), and actually moving the file pointer based on the calculated result.

In accordance with another embodiment of the present invention, there is provided a computer readable recording medium on which a program is recorded, the program being capable of moving a file pointer while automatically decrypting data block by block in kernel mode of a UNIX operating system when a system call to move the file pointer a second number of bytes in the negative direction from a predetermined byte location is invoked by a user process in a file in which data is encrypted by the block unit having a first number of bytes using a block encryption algorithm, wherein the block including a first area in which encrypted data is written, and a second area in which the number of bytes of effective data written in the first area is encrypted and written, the program comprising: (a) a program module capable of adjusting the present position of the file pointer to the predetermined byte location; (b) a program module capable of checking whether or not the position of the file pointer adjusted in the step (a) is the beginning of the corresponding block, and moving the file pointer to the beginning of the corresponding block according to the result; (c) a program module capable of continuing to repeat a linked list organizing procedure until the total added number of bytes of effective data of the organized linked list is equal to the second number of bytes requested by the user process, wherein the linked list organizing procedure performed block by block comprising a copying operation in which a block is copied to memory; a decrypting operation in which the copied block is

decrypted using the encryption key used when the file was encrypted; a linked list organizing operation in which the linked list for the decrypted block is organized in memory with reference to the number of bytes of effective data of the decrypted block, written in the second area of the block; and a file pointer moving operation in which the file pointer is moved by double the size of the block in the negative direction; and (d) a program module capable of calculating the actual number of bytes by which the file point must be moved actually by using the linked list obtained in the step (c), and actually moving the file pointer based on the calculated value.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagram schematically illustrating a structure of a file system of a UNIX operating system to which the present invention is applied;

FIG. 2A is a flow chart schematically illustrating a write system call (sys_write) procedure in kernel mode according to the present invention;

FIG. 2B is a diagram illustrating an example of a file formed by decrypting an encrypted 48-byte file according to the present invention;

FIG. 2C is a diagram illustrating an example of a linked list formed in memory by encrypting 40 bytes according to the present invention;

FIG. 2D is a diagram illustrating blocks in which operations of writing data of 40 bytes are performed according to the present invention when the present file pointer is positioned at the thirteenth location of the first block;

FIG. 3A is a flow chart schematically illustrating another example of a read system call (read_write) procedure in kernel mode according to the present invention;

FIG. 3B is a diagram illustrating an example of a file formed by decrypting an encrypted 48-byte file according to the present invention;

FIG. 3C is a diagram illustrating effective data when there is a 20 byte read request when a file pointer is at 13th byte position of a first block;

FIG. 3D is a diagram illustrating an example of a linked list formed in memory by decrypting 20 bytes according to the present invention;

FIG. 3E is a view diagram illustrating the content of a buffer to be returned to a user process;

FIG. 3F is a diagram illustrating an example of file pointer movements performed according to the present invention after the read system call occurred;

FIG. 4A is a flow chart schematically illustrating a file pointer moving system call (sys_lseek) procedure in kernel mode according to the present invention;

FIG. 4B is a diagram illustrating an example of a file pointer movement according to the present invention when an offset transferred from a user process has a positive value;

FIG. 4C is a diagram illustrating an example of a file pointer movement according to the present invention when an offset transferred from a user process has a negative value; and

FIG. 4D is a diagram illustrating a linked list formed in memory after a file was read block by block and decrypted while the file pointer is moved according to the present invention when an offset has positive value.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

The above objectives and advantages of the present invention will become more apparent by describing preferred embodiments thereof in detail with reference to the attached drawings. However, it should be understood that the preferred embodiments of the present invention described below are not intended to limit the scope of the present invention, but intended to describe the present invention to those skilled in the art more clearly and easily. The present invention is applicable to a general file as well as a security-classified

file. In the case of a general file, the steps derived from the reason why a system user has a level of security, i.e., the steps of inputting user's level of security when the user logs in a system, authenticating the level of security inputted by the user, checking the security level of a user process; checking whether an i-node (index-node) of a file is security-classified or not, and the like can be omitted. Here, the same reference numerals denote similar members in the drawings.

FIG. 1 shows components of a UNIX operating system taking a file system as a central portion so as to describe procedures of writing and reading data to/from a storage device in a UNIX operating system, and a procedure of moving a file pointer.

Referring to FIG. 1, when a process 10 of a general-user's application program writes a new file in user mode or modifies an existing file, a write system call, read system call, or a pointer moving system call which is a UNIX operating system call is performed, via system libraries, by a file system I including ext2 60, msdos 70, minix 80, proc 90, or the like supported as a virtual file system (VFS) 50. Here, the term "process 10" is used to generally indicate an application program which reads a file from a storage device 100 and writes a file to the storage device 100 such as a document producing program, a file processing program, an accounting program, and the like. The above system calls 20, 30, and 40 write data to or read data from the storage device 100, for example, a hard disk, diskette, or CD-ROM, and moves the position of a file pointer via a buffer cache of the file system I constituting kernel mode, and device drivers 120.

In the present invention, when data requiring security is written or read, the data is automatically and compulsively encrypted or decrypted while a user does not recognize whether or not the data is encrypted or decrypted by changing the procedures of performing the above three system calls using a block encryption algorithm (a detailed description will be given below).

Either a symmetric-key block encryption algorithm such as

SEED, DES, or 3-DES, or an asymmetric-key block encryption algorithm such as RSA may be used as the above block encryption algorithm. In the embodiments of the present invention, a SEED symmetric-key block encryption algorithm is employed, and the number of bits of an encryption key is 128.

In addition, in order to encrypt or decrypt data using the block encryption algorithm according to the present invention, a predetermined key is required. In the preferred embodiments of the present invention, keys are embedded in a kernel image when the kernel is compiled and generated, and then the embedded keys are utilized. Concretely, a system security manager lets a user decide a series of characters for generating a particular key. Then, a 128 bit seed encryption key is generated by using an encryption algorithm such as a hash encryption algorithm, for example, the MD5 (Message Digest 5) which can generate a predetermined-bit key, for example a 128 bit key by using the above series of characters. Thereafter, the seed encryption key is embedded in a kernel image when the kernel of the UNIX operating system is compiled. If the seed encryption key is embedded in the kernel image like this, whenever the system is booted, the seed encryption key is loaded together with the kernel image and stays resident in the kernel. At this time, it is preferable to employ a method in which after the seed encryption key is recorded in a smart card, the seed encryption key is inputted to the system by using the smart card when the kernel is compiled and generated.

In the present invention, the seed encryption key embedded in the kernel image of the UNIX operating system is used to automatically encrypt and decrypt a security-classified file in kernel mode, and this will be described in detail below when the preferred embodiments of the present invention is described.

Now, a method of automatically encrypting and decrypting data requiring security in kernel mode without respect to user's intention, and a method of moving a pointer a given number of bytes in an encrypted file will be described in sequence. In addition, in the embodiment which will be described below, the

size of a block for processing data by using the block encryption algorithm is set to be 16 bytes. However, it will be apparent to those skilled in the art that the size of a block is not limited to 16 bytes. For example, the size of a block to be used in encryption or decryption may be set to be 8 bytes, if necessary.

[A method of automatically encrypting data block by block in kernel mode, and storing the encrypted data to a storage device]

FIG. 2A shows a flow chart illustrating a series of operations occurring in kernel mode when a write system call (for example, sys_write) is made by a user process (step S1) in a UNIX operating system which the present invention is applied to. Here, the system call sys_write (step S1) is a kernel function used when a user process requests the kernel to do a write function service, and is referred to as a write system call hereinafter.

Referring to FIG. 2A, when a user process invokes a write system call via a library function such as write, fwrite, fprintf, putw, or the like, the kernel performs the job requested by the user process. For example, a function write (fd, buf, 40) is an example function to write 40 bytes of data stored in a buffer to a file designated by the file descriptor fd. When the write system call as described above exists, the i-node of the file is found first by using the file descriptor to get file information, and whether or not the system call is in writing mode, what memory device the data is written to, what file system is used during the process of performing the write system call, and the like are discriminated.

Thereafter, whether or not the data written to the memory device is the data that must be encrypted is decided (step S2). In performing the write system call modified according to the present invention, when whether or not the data to be written is data that must be encrypted is determined (step S2), the

determination is made compulsively in kernel mode, not according to the user's intention but according to whether or not the user process has a certain level of security. To this end, in the present invention, a security level field having predetermined bits is additionally assigned to the task structure of a user process. Then, user's level of security is written to the security level field in a predetermined manner, so that when whether or not data is encrypted is decided, the encryption of data can be compulsively controlled in kernel mode according to whether or not a certain level of security is written to the security level field. For example, when a security level field having 32 bits is additionally assigned to the task structure of a user process, and a portion of the field, for example, an 8 bit portion is set to be a security level writing portion, the number of levels of security can be 256. A method of writing the security level of a user process to the security level field includes a method in which user information including a security level is directly inputted by a user or indirectly inputted via a smart card by a user when the user log in the system. After the security level information inputted by the user is authenticated in an authenticating step based on a security level data base provided in advance in the system, the security level information is written to the security level field assigned in the task structure of the user process.

As described above, when the security level writing portion is set to have more than 1 bit, the number of levels of security can be more than 2. Anyhow, whether or not data is encrypted in kernel mode is decided depending on whether a certain level of security exists or not without respect to the type of security level.

Concretely, the security level of a process is found by a bitwise operation of the security level field assigned to the task structure of the process making a write system call. If the process requesting the write system call has no security level, the data requested to be written is classified as general data, and a memory area to which the data is written is locked using

the present position of a file pointer as a reference point. Then, requested bytes of the content of the buffer transferred from the user process is written to a file designated by the file descriptor fd (step S3). Thereafter, when the process completes the system call, the process returns to user mode (step S4). If, as a result of a bitwise operation of the security level field, the process making the write system call has a security level, the process branches to a block encryption routine according to the present invention. The procedures after branching is as follows.

In a method of encrypting data according to the present invention, since data is automatically encrypted block by block in kernel mode using a block encryption algorithm, whether or not the file pointer is presently positioned at the beginning point of a block must be checked (step 5). In block encryption, when the unit size of a block is 16 bytes, and when 16 cannot divide the number of bytes corresponding to the present position of the file pointer, it is determined that the present file pointer is not positioned at the beginning point of a block. When it is determined that the present file pointer is not positioned at the beginning point of a block, the file pointer is moved to the beginning point of the block (step S6). Then, the movement distance of the file pointer is stored in a stack.

In addition, in automatically encrypting data block by block in kernel mode, it is preferable that the number of bytes of effective data written to a block is stored in a particular byte location within the block, and the stored number is utilized when the data is decrypted. Therefore, in preferred embodiments according to the present invention, the unit size of a block is set to be 16 bytes, and the last byte is set to be an information recording location for recording the number of bytes of effective data. As a matter of course, when the size of a block is 16 bytes, any byte between the first byte and the sixteenth byte may be set to be an information recording location for recording the number of bytes of effective data.

FIG. 2B show a file to which data is to be written by a

write system call from a user process, and which is decrypted for the purpose of describing the present invention in an easy and convenient manner. Here, the file is decrypted by a method of automatically decrypting encrypted data in kernel mode according to the present invention, and the method will be described later. At the sixteenth byte locations of the first block A, the second block B, and the third block C, the numbers of bytes of effective data of respective blocks, i.e., 15, 7, and 11 are written. The eighth through fifteenth byte locations of the second block B, and the twelfth through fifteenth byte locations of the third block are null, which are the locations that data is not written to even though there is a write system call from a user process. In other words, since the numbers of bytes of effective data of the second and third blocks B and C are 7 and 11, respectively, writing data to the second and third blocks B and C beyond the numbers are not allowed.

Referring to FIGS. 2A and 2B, a method of automatically encrypting data in kernel mode according to the present invention will be described as follows. When the file pointer is presently positioned at the thirteenth byte location (in which 12 is written) of the first block A at the time when a write system call is made by a user process, the file pointer is moved to the first byte location (in which 0 is written), and the movement distance 12 is stored in the stack (step S6). In the present invention, since the data to be written is encrypted by the 16-byte block, and the sixteenth byte locations of blocks are assigned to the numbers of bytes of effective data of respective blocks, the number of blocks necessary for performing the write system call is found based on the number of bytes transferred from the user process (step S7). For example, when a write system call to write 40 bytes is made by a user process, and a file pointer is positioned at the thirteenth byte location of the first block A, since a linked list formed in memory, for example, a doubly linked list is configured to have the first block A in which the number of bytes of effective data is 15, and data is additionally written to only the thirteenth through

fifteenth byte locations, the second block B in which the number of bytes of effective data is 7, the third block C in which the number of bytes of effective data is 11, the fourth block (not shown) in which the number of bytes of effective data is 15, and the fifth block (not shown) in which the number of bytes of effective data is 4, the number of blocks to be encrypted is 5, and when the data is written to a memory device as a file, the data of five 16-byte blocks is encrypted block by block and stored in the memory device. When a new file is created, since a linked list formed in memory is configured to have three blocks in which the numbers of bytes of effective data are 15, 15, and 10, respectively, the number of blocks to be encrypted is 3, and when the data is written to a memory device as a file, the data of three 16-byte blocks is encrypted block by block.

An encryption key used to encrypt the data block by block may be the above-mentioned seed encryption key in itself. However, in the present invention, in order to realize more strengthened security of files, a unique key different from other file keys is generated based on the above seed encryption key and used for every file to be encrypted. To this end, when data of a security-classified file is encrypted, unique information concerning the classified file, for example, the time and date when the file is produced, the unique file number to designate the file or the like is extracted with reference to the i-node of the file. Then, the seed encryption key is encrypted again based on the information extracted from the i-node by using a predetermined encryption algorithm, for example, the SEED encryption algorithm, and accordingly a unique encryption key of a predetermined bits, for example, 128 bits is regenerated for the file.

In some cases, after data of predetermined bytes, for example, 16 bytes including user information such as user's password, and user's secret number, and unique file information such as the file number is produced, the seed encryption key of each file can be re-encrypted by using the above-mentioned 16-byte data. Thereafter, the re-encrypted key of a file is used as

an encryption key when data of the file is encrypted block by block according to the present invention. As described above, when encryption keys different from each other are used for individual security-classified files, more strengthened security can be realized.

After the number of blocks to be encrypted is calculated (step S7) as described above, a linked list of blocks is organized in memory (step S8) while the content of a buffer transferred from the user process is copied and encrypted by the numbers of bytes of effective data of respective blocks. At this time, encryption of individual bocks is performed via the encryption algorithm loaded in the kernel when the system is booted, and, to this end, when the kernel is compiled and produced, it is possible to cause the encryption algorithm to be embedded in the kernel. The method of embedding the encryption algorithm in the kernel can be contrived easily by those skilled in the art.

The final appearance of the linked list formed via the above step S8 is shown as a linked list diagram in FIG. 2c. The procedures of forming the linked list is as follows.

Referring to FIGS. 2A and 2C, while a linked list is organized block by block in memory (ⓐ → ⓑ → ⓒ →...→ ⓝ), the numbers of bytes of blocks in which a data is overwritten or newly written, are stacked in sequence (step S8). Then, whether or not the stacked number of bytes is less than the number of bytes requested by the user process is confirmed. If the stacked number of bytes is less than the number of bytes requested by the user process, the step S8 is repeated. For example, when the number of the organized blocks is less than the number of blocks calculated in the step S7 (step S9), the step S8 is repeated until a linked list ⓐⓑⓒ...ⓝ shown in FIG. 2C is formed.

If a write system call of writing 40 bytes is made by the user process in a data structure shown in FIG. 2B, the data portion is expressed in unencrypted form as shown in FIG. 2D. Shaded portions in FIG. 2D correspond to regions in which data is overwritten or newly written, and the sixteenth location of

each block is a region in which the number of bytes of effective data of each block is written.

After a linked list shown in FIG. 2C is organized in memory during the steps S8 and S9, whether or not a linked list exists in memory is confirmed (step S10). That is, in the linked list obtained as a result of the operations in the steps S8 and S9, whether the value of a pointer indicating the first linked list showed in FIG. 2C is null, is confirmed (step S10). When the value of the pointer indicating the first linked list is found not to be null in the step S10, the linked list ⓐ ⓑ ⓒ ... ⓝ organized in memory is written to a storage device by the 16 bytes sequentially (step S12). While the Step S12 is performed, whether or not the stacked number of written bytes is less than the number of bytes requested to be written by the user process is determined (step S13). The step S12 is repeated until the stacked number is equal to or larger than the requested number by the user process. When the stacked number is equal to or larger than the requested number by the user process, the level of security of the user process is copied to a reserved field that is an unused field of the i-node of the file. To this end, a security level field having a size of predetermined bits, for example, 32 bits is additionally assigned to the reserved field of the i-node, and the security level of the user process is copied to the security level field. The MSB (Most Significant Bit) which is an encryption flag having a predetermined number of bits, for example, 1 bit is assigned to the security level field, and when writing the linked list to the storage device (steps S12 and S13) is completed, the MSB is subjected to a bitwise OR operation performed with 1, and can be set to indicate that the file is an encrypted file (step S14). After the step 14 is completed, all the assigned system resources are released, and the process returns to user mode.

[A method of automatically decrypting a file encrypted according to the present invention and stored in a storage device, block by block in kernel mode]

FIG. 3A show a flow chart illustrating a series of operations occurring in kernel when a read system call (for example, sys_read) is made by a user process (step S16) in the UNIX operating system which the present invention is applied to. Here, sys_read (step S16) is a kernel function used when a user process requests the kernel to do a read function service, and is referred to as a read system call hereinafter. In addition, a method of automatically decrypting data in kernel mode, which will be described below, is applied to a file which is encrypted by a method of automatically encrypting data in kernel mode according to the present invention, described with reference to FIGS. 2A through 2D, and which is stored in a storage device.

Referring to FIG. 3A, when a user process invokes a read system call via a system library function such as read, fread, fscanf, getw, getc, or the like, the kernel performs the job requested by the user process. For example, a function read (fd, buf, 20) is an example function to read 20 bytes of data from a file designated by the file descriptor fd and to store the data in a buffer. In a method of automatically decrypting data in kernel mode according to the present invention, in order to determine whether or not a file to read is an encrypted file (step S17), the i-node of the file is found by using the file descriptor fd transferred from the user process, and the MSB of the security level field included in the i-node is subjected to a bitwise AND operation performed with 1.

When the file designated by the file descriptor fd is determined not to be an encrypted file in the step S17, the file is classified as a general file. Thereafter, the data which are to be read is locked regarding the present position of a file pointer as a reference point, and then the requested number of bytes of data is read from a storage device, and then the read content is stored in the buffer transferred from the user process (step S18). When the process completes the read system call, the process returns to user mode (step S19).

The file designated by the file descriptor fd is determined

to be an encrypted file in the step S17, the process branches to a decryption routine according to the present invention. In a method of decrypting data in kernel mode according to the present invention, since data to be decrypted was encrypted by the automatic block encryption method described with reference to FIG. 2A through 2D, the data is decrypted block by block. In addition, when a block of 16 bytes is set to be a unit size of encryption, a block of 16 bytes is set to be a unit size of decryption in a method of automatically decrypting data in kernel mode according to the present invention.

Therefore, in order to decrypt the whole encrypted file or a selected portion of the file, whether or not the present file pointer is positioned at the beginning of a block must be checked (step S20). In the checking method, when 16 cannot divide the number of bytes corresponding to the present position of the file pointer, it is determined that the present file pointer is not positioned at the beginning of a block. In the step S20, when 16 cannot divide the number of bytes corresponding to the present position of the file pointer, the file pointer is moved to the beginning point of the block (step S21). Then, the movement distance of the file pointer to the beginning point of the block is saved in a stack.

FIG. 3B shows a diagram illustrating a file decrypted in advance for the convenience of describing the present invention. In an embodiment of a method of automatic decrypting data according to the present invention, on the condition that a read system call to read 20 bytes data is made by a user process, and the present file pointer is positioned at the thirteenth byte location of the first block A, a series of operations of decrypting the data stored in the region of the thirteenth through fifteenth bytes of the first block A, in the region of the first through seventh bytes of the second block B, and in the region of the first through tenth bytes of the third block C is exemplified.

Referring to FIGS. 3A and 3B, since the present file pointer is not positioned at the beginning point (the first byte

location in which 0 is written) of the first block A, first, the file pointer is moved to the beginning point of the first block A, and the number 12 which is the movement distance thereof is stored in a stack. Then, after the blocks are respectively read and decrypted in sequence (step S22), effective data of the blocks is organized as a linked list and stored in memory with reference to the sixteenth byte of each block (step 22), and, at the same time, the numbers of effective data to be read are stacked one by one (step 22). At this time, a decryption key used to decrypt each block is a key used when the data to be decrypted was encrypted using a symmetric key encryption algorithm such as SEED.

FIG. 3C shows a diagram indicating that when the read system call to read 20 bytes data from the file having the data structure shown in FIG. 3B is made by the user process, which regions of data is to be decrypted according to the present invention. In FIG. 3C, the regions of data to be decrypted by the present invention are displayed in shaded form.

Referring to FIG. 3C, it can be found that regions of the thirteenth through fifteenth bytes of the first block A, the first through seventh bytes of the second block B, and the first through tenth bytes, i.e., only the shaded regions are selectively decrypted.

FIG. 3D shows a linked list organized in memory while data is decrypted block by block in sequence according to the present invention. The procedure of organizing the linked list is as follows. First, after the first block is decrypted, a first portion @ of the linked list (the block A) is organized in memory. While each block is decrypted and the linked list thereof is organized in memory in sequence with reference to the sixteenth byte location, the numbers of bytes to be read in decrypted blocks are stacked. In block A, the numbers of bytes to be read can be calculated based on the movement number of file pointer in Step S21. When the stacked number of bytes is less than the number of bytes requested to be read by the user process, the procedure of organizing the linked list is repeated

concerning the second, third, ... blocks, so the linked list ⓐⓑⓒ shown in FIG. 3D can be organized in memory.

Subsequently, in the linked list of blocks organized in the steps S22 and S23, whether the value of a pointer indicating the first linked list ⓐ is null is confirmed (step S24). When the value of the pointer indicating the first linked list ⓐ is found not to be null in the step S24, the effective data(refer to shaded area in FIG. 3C) of the linked list ⓐⓑⓒ organized in memory is copied to the buffer transferred from the user process sequentially (steps S26 and S27). The operation of copying the effective data to be read from the linked list to the buffer transferred from the user process is continued until the number of bytes of copied data is equal to the number of bytes requested to be read by the user process (step S27). After the operation of copying the effective data of the linked list to the buffer is completed, the position of the file pointer is adjusted (step 28), and then the process returns to user mode. At this time, the content of the buffer is as shown in FIG. 3E. On the other hand, when the value of the pointer indicating the first linked list is found to be null in the step S24, the process returns to user mode (step 25).

FIG. 3F shows a movement sequence of the file pointer in the operations performed from steps S20 to S28 using circled letters. Concretely, when a system call to read 20 bytes of data from a file designated by the file descriptor fd and to store the data in a buffer is made by the user process, the file pointer is positioned at the thirteenth byte location (please refer to ① in FIG. 3F) of the first block A. However, since the file pointer is not positioned at the beginning point of the block, the file pointer is moved to the beginning point of the block (please refer to ②). After the first block A is read (the file pointer moves to the beginning point (please refer to ③) of the second block automatically) and then decrypted, a portion of the linked list (please refer to ⓐ in FIG. 3D) corresponding to the first block A is organized in memory with reference to the number of bytes of effective data recorded in the sixteenth byte

location. Subsequently, after the second block B is read (the file pointer moves to the beginning point (please refer to ④) of the third block automatically) and then decrypted, a portion of the linked list (please refer to ⓑ in FIG. 3D) corresponding to the second block B is organized in memory with reference to the number of bytes of effective data recorded in the sixteenth byte location of the block B. Since the added number of bytes of effective data(substantially to be read) of the linked list, i.e., 10 is less than the number of bytes requested by the user process, i.e., 20, the procedure of decrypting the third block continues. As described above, after the third block C is read (the file pointer moves to the beginning point (please refer to ⑤) of the fourth block automatically)and then decrypted, a portion of the linked list (please refer to ⓒ in FIG. 3D) corresponding to the first block C is organized in memory with reference to the number of bytes of effective data recorded in the sixteenth byte location of the block C. Since the added number of bytes of effective data(substantially to be read) is 10 when the portions ⓐ and ⓑ of the linked list are organized, 10 bytes among 13 bytes of data of the third block C is used to organize a portion ⓒ of the linked list. After the portion ⓒ of the linked list is organized, again, the file pointer is moved to the eleventh byte location ⑥ of the third block C. Finally, the kernel releases all the assigned resources, and the process returns to user mode (step 29).

In the below, as data of a security-classified file is encrypted and decrypted automatically in kernel mode according to the present invention, how a file pointer moving system call procedure for moving the file pointer predetermined bytes is modified and performed will be described with reference to FIGS. 4A through 4D.

FIG. 4A shows a flow chart illustrating a file pointer moving system call (sys_lseek) (step S30) procedure in kernel mode, modified according to the present invention. Here, sys_lseek (step S30) generally designates system call functions used when a user process requests that the kernel do a file

pointer moving service.

Referring to FIG. 4A, when a user process makes a system call via a system library function such as fseek, lseek, or the like, the kernel performs a job requested by the user process. For example, lseek(fd, offset, origin) is a kernel function for moving a file pointer by the bytes corresponding to offset bytes from origin which is the original file pointer position of a file designated by a file descriptor fd.

In a file pointer moving system call modified according to the present invention, whether or not the file to be read is encrypted can be determined (step S31) by finding the i-node of the file using the transferred file descriptor, and performing a bitwise AND operation with 1 to the MSB of the security level field in the i-node.

In the step S31, if it is decided that the file designated by the file descriptor fd is not an encrypted file, the file is classified as a general file. Then, the file pointer is moved by offset bytes from "origin" transferred from the user process(step S32). Thereafter, when the process completes the file pointer moving system call, the process returns to user mode (step S33).

In the step S31, if it is decided that the file designated by the file descriptor fd is a file automatically encrypted according to the present invention, the step (step S34) of adjusting the position of the file pointer according to the value of origin transferred from the user process is performed. For example, when the value of origin transferred from the user process is 2, the file pointer is moved to the end of the file, when the value is 0, the file pointer is moved to the beginning of the file, and when the value is 1, the file pointer is maintained at the present position. In addition, a file automatically encrypted according to the present invention, is characterized in that the number of bytes of effective data recorded in each block can be found when each block is read and decrypted, and then the sixteenth byte location of each block is referred to. Therefore, after the step S34, when it is decided

that the position of the file pointer is the end of the file, the position of the file pointer is moved by the size of a block in the negative direction. For example, when the size of a block is 16 bytes, the position of the file pointer is moved by 16 bytes in the negative direction.

Thereafter, whether or not the file is normally encrypted is checked based on the fact that whether or not 16 can divide the size of the file (step 35). In the step S35, if 16 divide the size of file so that a remainder does not yield, the file is considered as being normally encrypted by the present invention. In this case, taking into account that the number of effective data of each block is written at the last byte location of the block, each block is read and decrypted, and then the effective data of the block is organized into a linked list and stored in memory, with reference to the sixteenth byte location of each block. As a matter of course, the numbers of effective data of each block are stacked during the process of organizing the linked list(step S37). At this time, the above-mentioned decryption process can be performed with reference to the preferred embodiment of a method of automatically decrypting method in kernel mode according to the present invention, described with reference to FIGS. 3A through 3D.

In the step S35, if 16 divide the size of the file so that a remainder yields as a result of calculation, since the file is not normally encrypted according to the present invention, the process returns to user mode (step S36).

Meanwhile, in executing the step S37, methods of reading each block vary with the offset value transferred from the user process. Concretely, when the offset value is a positive integer, data of each block is read and decrypted sequentially. However, in the case that the offset value is a negative integer, after 16 bytes of data are read, it is necessary to adjust the position of the file pointer by 32 bytes in the negative direction. This will become more apparent by the following description with reference to FIG. 4B showing an example of movements of the file pointer in the case of +offset, and FIG.

4C showing another example of movements of the file pointer in the case of -offset.

Referring to FIG. 4B, when the user process makes a request to move the file pointer +20 bytes when the file pointer is positioned at the ① location of the first block A, first, the file pointer is moved to the ② location of the first block A. Then, after the first block A is read (the file pointer is automatically moved to the ③ location) and decrypted, a first portion of a linked list is organized in memory while the number of bytes of effective data is added to an initial value, i.e., 0. Similarly, after the second block B is read (the file pointer is automatically moved to the ④ location) and decrypted, a second portion of the linked list is organized in memory while the number of bytes of effective data is cumulatively added to the previously added value. Similarly, after the third block C is read (the file pointer is automatically moved to the ⑤ location) and decrypted, a third portion of the linked list is organized in memory while the number of bytes of effective data is cumulatively added to the previously added value. This block-by-block procedure is repeated if the total number of bytes of effective data is less than the offset, i.e., 20 (step S38). Therefore, when the third portion of the linked list including data from the first byte to the eleventh byte (the number of effective bytes of the first block A, i.e., 3 + the number of effective bytes of the second block B, i.e., 7 = 10) of the third block is organized in memory, the step 37 is completed.

Referring to FIG. 4C, when the file pointer is requested to move backward by 20 bytes when the file pointer is positioned at the ① location of the third block, first, the file pointer is moved to the ② location of the third block C. Then, after the third block C is read (the file pointer is automatically moved to the ③ location of the fourth block D) and decrypted, a first portion of a linked list is organized in memory while the number of effective bytes of effective data, i.e., 10 is added to an initial value, i.e., 0. Then, after the file pointer is moved to the ④ location of the second block B (the file pointer is

adjusted to move -32 bytes), and the second block B is read (the file pointer is automatically moved to the ⑤ location of the third block) and decrypted, a second portion of the linked list is organized in memory while the number of bytes of effective data, i.e., 7 is cumulatively added to the previously added value. Similarly, after the file pointer is moved to the ⑥ location of the first block A (the file pointer is adjusted to move backward by 32 bytes), and the first block A is read (the file pointer is automatically moved to the ⑦ location of the second block B) and decrypted, a third portion of the linked list is organized in memory while the number of bytes of effective data is cumulatively added to the previously added value. This block-by-block procedure is repeated if the total number of bytes of effective data is less than the offset, i.e., 20.

FIG. 4D shows an example of the linked list organized in memory after data is read and decrypted block by block, by the above-described method.

Referring to FIGS. 4A and 4D, when the numbers of bytes of effective data of blocks cumulatively added while the linked list is organized block by block in memory, is less than the offset transferred from the user process (step S38), the above block-by-block procedure (step S37) is repeated to organize the linked list ⓐⓑⓒ...ⓝ (step S37).

Thereafter, in the result of the steps S37 and S38, whether or not the value of a pointer indicating the first linked list is null, is confirmed (step S39). When the value of the pointer indicating the first linked list is found not to be null in the step S39, the node of the linked list organized in memory is combined, and the number of bytes of effective data is cumulatively added (step S41). Then, whether or not the added number is less than the offset transferred from the user process is confirmed in step S42. The step S41 is repeated until both the values of the added number and the offset are the same.

After steps S41 and S42 is executed, taking into consideration the remaining portions other than the effective

data regions in the 16 bytes encrypted blocks, an actual offset for the file pointer to move during the process of the file pointer movement, is found in step S43. In the case of the example of movements of a file pointer shown in FIG. 4B, when a request to move the file pointer by +20 offset bytes is made by a file pointer moving system call, the file pointer is moved from the thirteenth location of the first block A (the ① location) to the eleventh location of the third block C (the ⑥ location). However, since the actual offset by which the file pointer is moved, must be decided by taking account of the eighth through fifteenth locations of the second block that have a null value, the actual offset by which the file pointer is moved is +28 bytes.

When the actual offset found in the step 43 is null, the process returns to user mode (step S44), and when the actual offset found in the step 43 is not null, the file pointer is moved by the actual offset founded in step S45. After the step S45 is executed, all the assigned resources are released, and the process returns to user mode (step S46).

The method of automatically encrypting and decrypting a file in kernel mode, and the method of moving a file point using the same which are described above according to the present invention can be programmed and recorded in a computer readable recording medium. At this time, the flowcharts shown in FIGS. 2A, 3A, and 4A can be used as algorithms when the technology applied to the present invention is programmed. It should be understood that the methods according to the present invention can be easily programmed by those skilled in the art when the flow charts disclosed in the attached drawings and relative descriptions are provided to them.

Although preferred embodiments of the present invention have been described in detail with reference to the accompanying drawings, it should be understood that various modifications and improvements may be made by those skilled in the art without departing from the spirit and scope of the invention.

According to one aspect of the present invention, since

only necessary partial data requested by a user process is encrypted or decrypted in kernel mode using a linked list, data can be processed at a speed higher than that of a conventional method of encrypting or decrypting a file in user mode.

According to another aspect of the present invention, data is encrypted or decrypted in kernel mode in a state strictly separated from an address space of a user application program. And, data is stored in a storage device after encrypted, not in a manner in which a user having a level of clearance selects as to whether or not the data is encrypted, but in a compulsive manner in kernel mode. So that, security can be assured even though the storage device is illegally stolen or the content of the storage device is illegally copied.

According to still another aspect of the present invention, since a system security manager manages encryption key, management of keys of a general user can be simple.

What is claimed is:

1. A method of automatically encrypting, in kernel mode of a UNIX operating system, wherein a second number of bytes of data requested by a user process to be newly written by the block unit having a first number of bytes and including a first area in which data is actually written, and a second area in which the number of bytes of the data written in the first area is recorded, comprising the steps of:

(a) calculating the number of blocks required for encrypting the second number of bytes of data while taking into consideration the number of bytes constituting the first area and the second number of bytes of data;

(b) copying a proper portion of data requested to be written from a buffer transferred from the user process to the first area, writing the number of bytes of the data written in the first area to the second area, encrypting the data written in the first and second areas by the unit of the first number of bytes using a predetermined encryption algorithm, and organizing a linked list of corresponding blocks in memory, sequentially with respect to individual blocks, wherein the number of the blocks is calculated in the step (a); and

(c) writing the linked list of the blocks organized in memory to a storage device.

2. The method as claimed in claim 1, wherein, before the step (b), the method further comprises the step of checking whether or not the present file pointer is positioned at the beginning of a block, and moving the file pointer to the beginning of the block depending on the result.

3. The method as claimed in claim 1, wherein the second area is an area used to record the number of bytes of data written in the first area, and is allocated at a predetermined

location in the block.

4. The method as claimed in claim 1, wherein the number of blocks to be encrypted in the step (a) is calculated by dividing the second number of bytes of data by the number of bytes of the first area, and adding 1 to the integer quotient thereof.

5. The method as claimed in claim 1, wherein after the step (c), the method further comprises the step of writing an encryption flag in a security level field allocated in advance in a reserved field of the i-node of an encrypted file to indicate that the file is encrypted.

6. The method as claimed in claim 1, wherein the encryption algorithm used in the step (b) is a block encryption algorithm.

7. The method as claimed in claim 6, wherein an encryption key used in the block encryption algorithm is a key embedded in a kernel image when the kernel is compiled and generated.

8. The method as claimed in claim 7, wherein the encryption key is a seed encryption key having predetermined bits, which is produced by letting a user determine a series of unique letters for generating a key and then encrypting the series of unique letters by a predetermined encryption algorithm.

9. The method as claimed in claim 8, wherein whenever a write system call is invoked, the seed encryption key is re-encrypted using information which is unique to a file and is included in an i-node of the file, and then the re-encrypted seed encryption key is used as an encryption key in the block encryption algorithm of the step (b).

10. The method as claimed in claim 1, wherein before the step (a), the method further comprises the step of:

(a1) letting a user input user information including security level information when the user logs in a UNIX system, and authenticating the user;

(b1) writing the user's security level information in a security level field, having predetermined bits and allocated in a task structure of a user process when the user is authenticated in the step (a1); and

(c1) determining whether or not the data requested to be written is data which must be encrypted by confirming whether or not security level information is written in the security level field of the task structure.

11. The method as claimed in claim 10, wherein, after the step (c1), the method further comprises the step of copying the user's security level information written in the task structure of the user process and writing the copied security level information in a reserved field allocated in a reserved field of the i-node of the file in which the data is encrypted and stored.

12. A method of automatically decrypting data block by block in kernel mode of a UNIX operating system when a system call to read a second number of bytes of data from a file which is encrypted by the block unit having a first number of bytes using a block encryption algorithm, and to store the data in a buffer is invoked, wherein the block including a first area in which encrypted data is written, and a second area in which the number of bytes of effective data written in the first area is encrypted and written, the method comprising the steps of:

(a) adjusting the position of a file pointer to the beginning of a block by taking the time when the write system call is invoked as a reference point;

(b) continuing to repeat a linked list organizing procedure until the total added number of bytes of effective data of the organized linked list is equal to the second

number of bytes requested to read, wherein the linked list organizing procedure performed block by block comprising a copying operation in which a block is copied to memory; a decrypting operation in which the copied block is decrypted using the encryption key used when the file was encrypted; and a linked list organizing operation in which the linked list for the decrypted block is organized in memory with reference to the number of bytes of effective data of the decrypted block, written in the second area of the block ; and

(c) copying the linked list obtained in the step (b) to the buffer.

13. A method of moving a file pointer while automatically decrypting data block by block in kernel mode of a UNIX operating system when a system call to move the file pointer a second number of bytes in the positive direction from a predetermined byte location is invoked by a user process in a file in which data is encrypted by the block unit having a first number of bytes using a block encryption algorithm, wherein the block including a first area in which encrypted data is written, and a second area in which the number of bytes of effective data written in the first area is encrypted and written, the method comprising the steps of:

(a) adjusting the present position of the file pointer to the predetermined byte location;

(b) checking whether or not the position of the file pointer adjusted in the step (a) is the beginning of the corresponding block, and moving the file pointer to the beginning of the corresponding block according to the result;

(c) continuing to repeat a linked list organizing procedure until the total added number of bytes of effective data of the organized linked list is equal to the second number of bytes requested from the user process,

wherein the linked list organizing procedure performed block by block comprising a copying operation in which a block is copied to memory; a decrypting operation in which the copied block is decrypted using the encryption key used when the file was encrypted; and a linked list organizing operation in which the linked list for the decrypted block is organized in memory with reference to the number of bytes of effective data of the decrypted block, written in the second area of the block; and

(d) calculating the actual number of bytes by which the file point must be moved actually by using the linked list obtained in the step (c), and actually moving the file pointer based on the calculated result.

14. A method of moving a file pointer while automatically decrypting data block by block in kernel mode of a UNIX operating system when a system call to move the file pointer a second number of bytes in the negative direction from a predetermined byte location is invoked by a user process in a file in which data is encrypted by the block unit having a first number of bytes using a block encryption algorithm, wherein the block including a first area in which encrypted data is written, and a second area in which the number of bytes of effective data written in the first area is encrypted and written, the method comprising the steps of:

(a) adjusting the present position of the file pointer to the predetermined byte location;

(b) checking whether or not the position of the file pointer adjusted in the step (a) is the beginning of the corresponding block, and moving the file pointer to the beginning of the corresponding block according to the result;

(c) continuing to repeat a linked list organizing procedure until the total added number of bytes of effective data of the organized linked list is equal to the second

number of bytes requested by the user process, wherein the linked list organizing procedure performed block by block comprising a copying operation in which a block is copied to memory; a decrypting operation in which the copied block is decrypted using the encryption key used when the file was encrypted; a linked list organizing operation in which the linked list for the decrypted block is organized in memory with reference to the number of bytes of effective data of the decrypted block, written in the second area of the block; and a file pointer moving operation in which the file pointer is moved by double the size of the block in the negative direction; and

(d) calculating the actual number of bytes by which the file point must be moved actually by using the linked list obtained in the step (c), and actually moving the file pointer based on the calculated result.

15. A computer readable recording medium on which a program is recorded, the program being capable of automatically encrypting, in kernel mode of a UNIX operating system, a second number of bytes of data requested by a user process to be newly written by the block unit having a first number of bytes and including a first area in which data is actually written, and a second area in which the number of bytes of the data written in the first area is recorded, the program comprising:

(a) a program module capable of calculating the number of blocks required for encrypting the second number of bytes of data while taking into consideration the number of bytes constituting the first area and the second number of bytes of data;

(b) a program module capable of copying data requested to be written from a buffer transferred from the user process to the first area, and writing the number of bytes of the data written in the first area to the second area, encrypting, by the first number of bytes,

the data written in the first and second areas using a block encryption algorithm, and organizing a linked list of corresponding block in memory, with respect to individual blocks, wherein the number of blocks for data to be written and encrypted is calculated by the program module (a); and

(c) a program module capable of writing the linked list of the blocks organized in memory to a storage device.

16. A computer readable recording medium on which a program is recorded, the program being capable of automatically decrypting data block by block in kernel mode of a UNIX operating system when a system call to read a second number of bytes of data from a file which is encrypted by the block unit having a first number of bytes using a block encryption algorithm, and to store the data in a buffer is invoked, wherein the block including a first area in which encrypted data is written, and a second area in which the number of bytes of effective data written in the first area is encrypted and written, the program comprising:

(a) a program module capable of adjusting the position of a file pointer to the beginning of a block by taking the time when the write system call is invoked as a reference point;

(b) a program module capable of continuing to repeat a linked list organizing procedure until the total added number of bytes of effective data of the organized linked list is equal to the second number of bytes requested to read, wherein the linked list organizing procedure performed block by block comprising a copying operation in which a block is copied to memory; a decrypting operation in which the copied block is decrypted using the encryption key used when the file was encrypted; and a linked list organizing operation in which the linked list for the decrypted block is organized in memory with reference to the number of

bytes of effective data of the decrypted block, written in the second area of the block; and

(d) a program module capable of copying the linked list obtained in the step (b) to the buffer.

17. A computer readable recording medium on which a program is recorded, the program being capable of moving a file pointer while automatically decrypting data block by block in kernel mode of a UNIX operating system when a system call to move the file pointer a second number of bytes in the positive direction from a predetermined byte location is invoked by a user process in a file in which data is encrypted by the block unit having a first number of bytes using a block encryption algorithm, wherein the block including a first area in which encrypted data is written, and a second area in which the number of bytes of effective data written in the first area is encrypted and written, the program comprising:

(a) a program module capable of adjusting the present position of the file pointer to the predetermined byte location;

(b) a program module capable of checking whether or not the position of the file pointer adjusted in the step (a) is the beginning of the corresponding block, and moving the file pointer to the beginning of the corresponding block according to the result;

(c) a program module capable of continuing to repeat a linked list organizing procedure until the total added number of bytes of effective data of the organized linked list is equal to the second number of bytes requested by the user process, wherein the linked list organizing procedure performed block by block comprising a copying operation in which a block is copied to memory; a decrypting operation in which the copied block is decrypted using the encryption key used when the file was encrypted; and a linked list organizing operation in which the linked list for the

decrypted block is organized in memory with reference to the number of bytes of effective data of the decrypted block, written in the second area of the block; and

(d) a program module capable of calculating the actual number of bytes by which the file point must be moved actually by using the linked list obtained in the step (c), and actually moving the file pointer based on the calculated result.

18. A computer readable recording medium on which a program is recorded, the program being capable of moving a file pointer while automatically decrypting data block by block in kernel mode of a UNIX operating system when a system call to move the file pointer a second number of bytes in the negative direction from a predetermined byte location is invoked by a user process in a file in which data is encrypted by the block unit having a first number of bytes using a block encryption algorithm, wherein the block including a first area in which encrypted data is written, and a second area in which the number of bytes of effective data written in the first area is encrypted and written, the program comprising:

(a) a program module capable of adjusting the present position of the file pointer to the predetermined byte location;

(b) a program module capable of checking whether or not the position of the file pointer adjusted in the step (a) is the beginning of the corresponding block, and moving the file pointer to the beginning of the corresponding block according to the result;

(c) a program module capable of continuing to repeat a linked list organizing procedure until the total added number of bytes of effective data of the organized linked list is equal to the second number of bytes requested by the user process, wherein the linked list organizing procedure performed block by block

comprising a copying operation in which a block is copied to memory; a decrypting operation in which the copied block is decrypted using the encryption key used when the file was encrypted; a linked list organizing operation in which the linked list for the decrypted block is organized in memory with reference to the number of bytes of effective data of the decrypted block, written in the second area of the block; and a file pointer moving operation in which the file pointer is moved by double the size of the block in the negative direction; and

(d) a program module capable of calculating the actual number of bytes by which the file point must be moved actually by using the linked list obtained in the step (c), and actually moving the file pointer based on the calculated result.

1/12

FIG.1

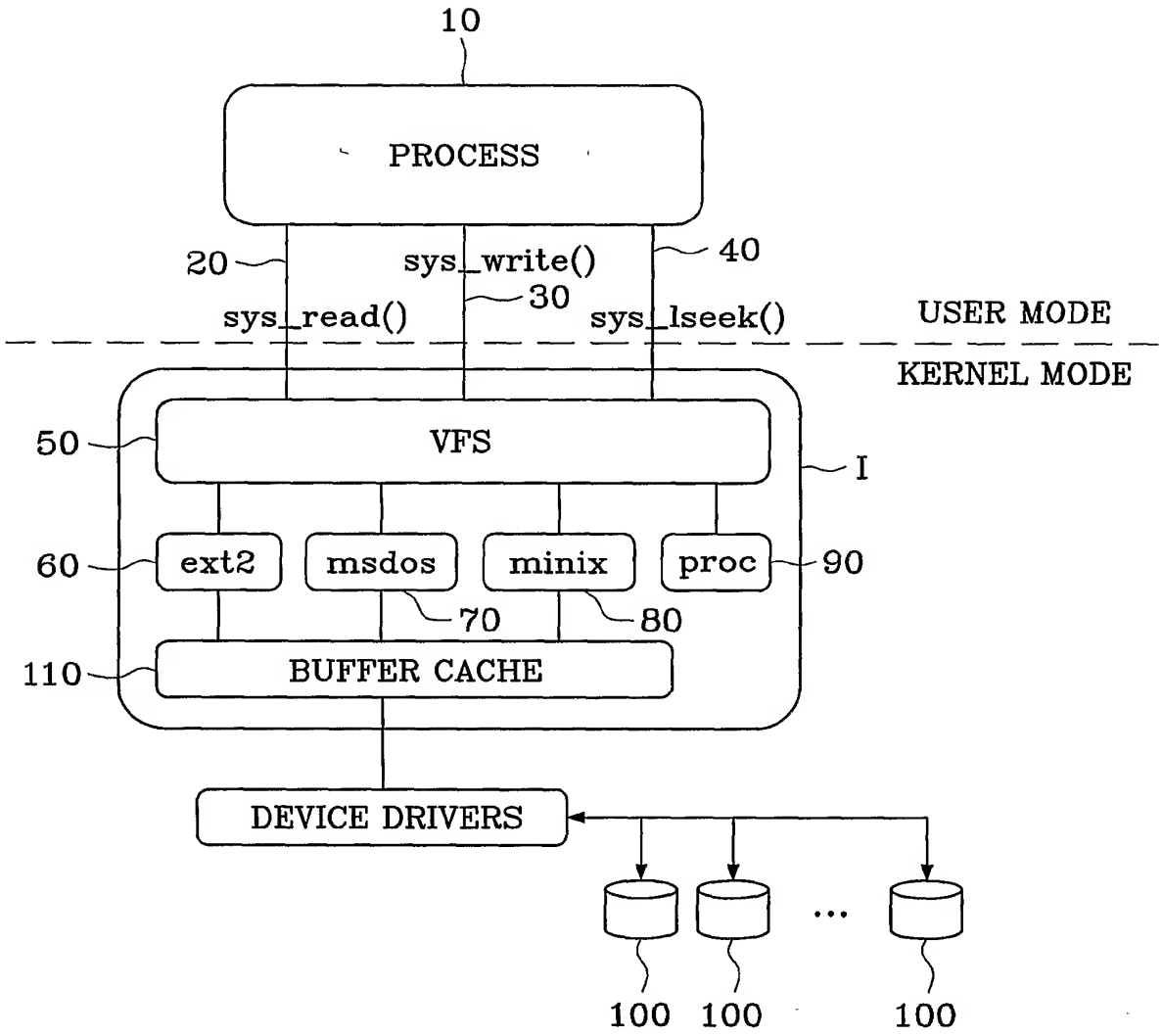

10
PROCESS
20
sys_write()
40
30
sys_read()
sys_lseek()
USER MODE
KERNEL MODE
50
VFS
I
60
ext2
msdos
minix
proc
90
70
80
110
BUFFER CACHE
DEVICE DRIVERS
...
100
100
100

FIG.2A

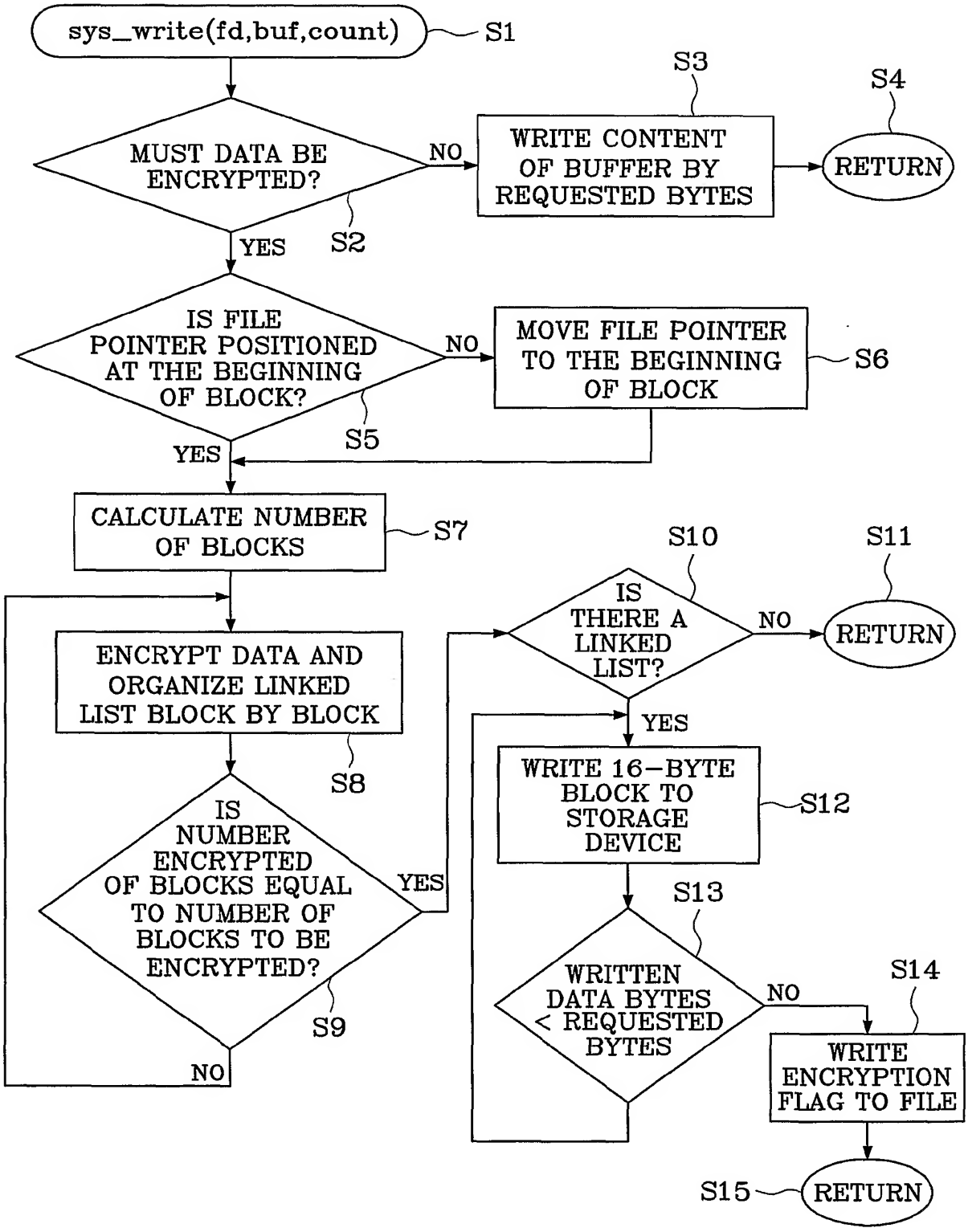

sys_write(fd,buf,count)
S1
MUST DATA BE ENCRYPTED?
NO
S2
YES
WRITE CONTENT OF BUFFER BY REQUESTED BYTES
S3
RETURN
S4
IS FILE POINTER POSITIONED AT THE BEGINNING OF BLOCK?
NO
S5
YES
MOVE FILE POINTER TO THE BEGINNING OF BLOCK
S6
CALCULATE NUMBER OF BLOCKS
S7
ENCRYPT DATA AND ORGANIZE LINKED LIST BLOCK BY BLOCK
S8
IS NUMBER ENCRYPTED OF BLOCKS EQUAL TO NUMBER OF BLOCKS TO BE ENCRYPTED?
YES
S9
NO
IS THERE A LINKED LIST?
S10
NO
RETURN
S11
YES
WRITE 16-BYTE BLOCK TO STORAGE DEVICE
S12
WRITTEN DATA BYTES < REQUESTED BYTES
S13
NO
WRITE ENCRYPTION FLAG TO FILE
S14
RETURN
S15

FIG.2B

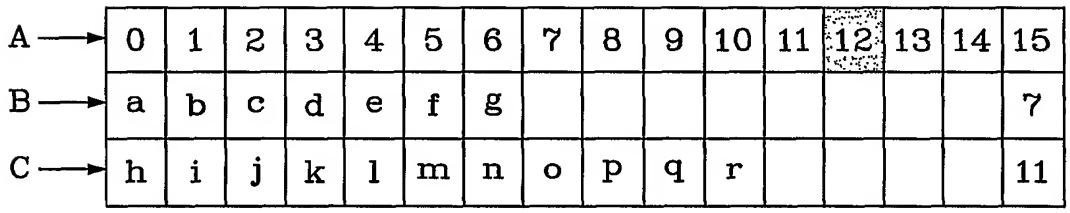


FIG.2C

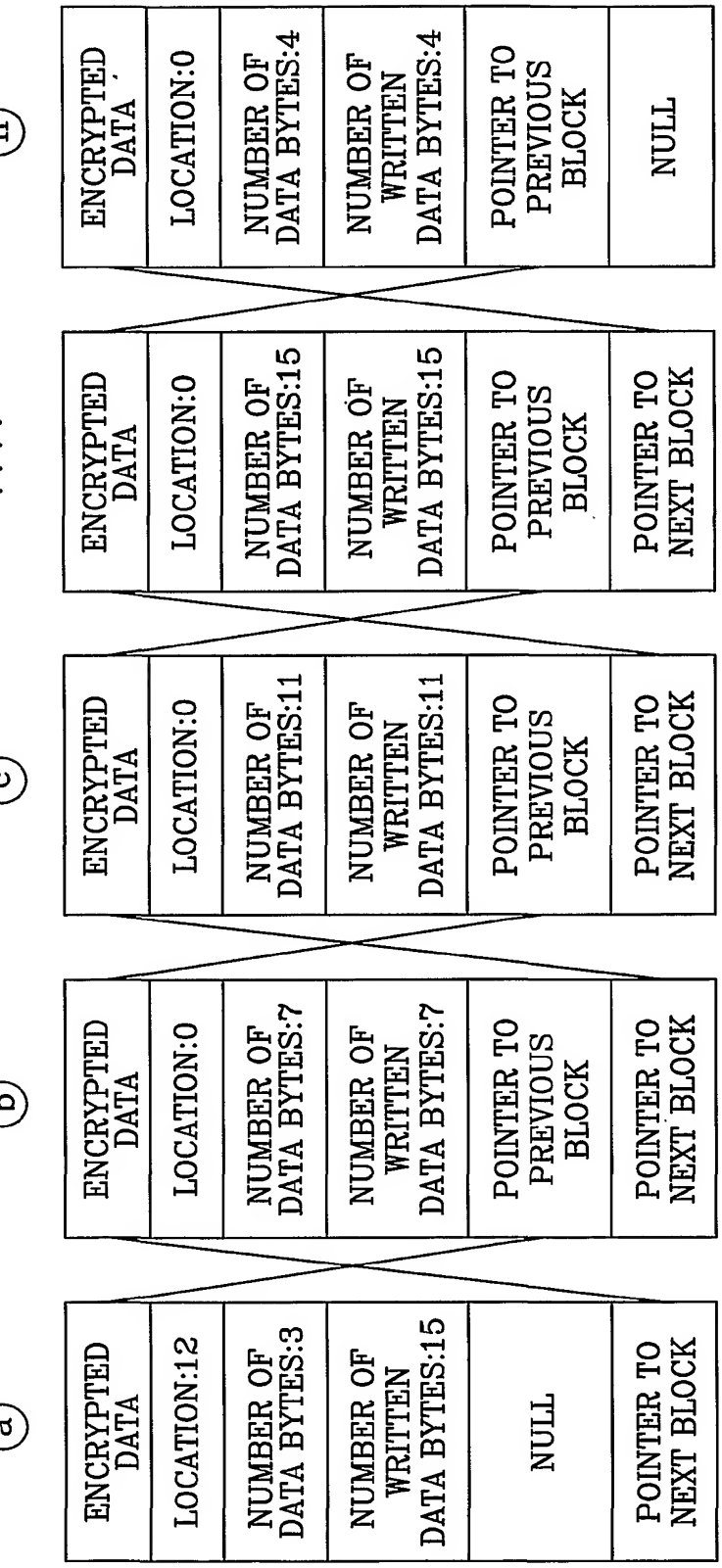

ⓐ
ENCRYPTED DATA
LOCATION:12
NUMBER OF DATA BYTES:3
NUMBER OF WRITTEN DATA BYTES:15
NULL
POINTER TO NEXT BLOCK
ⓑ
ENCRYPTED DATA
LOCATION:0
NUMBER OF DATA BYTES:7
NUMBER OF WRITTEN DATA BYTES:7
POINTER TO PREVIOUS BLOCK
POINTER TO NEXT BLOCK
ⓒ
ENCRYPTED DATA
LOCATION:0
NUMBER OF DATA BYTES:11
NUMBER OF WRITTEN DATA BYTES:11
POINTER TO PREVIOUS BLOCK
POINTER TO NEXT BLOCK
.....
ⓝ
ENCRYPTED DATA
LOCATION:0
NUMBER OF DATA BYTES:15
NUMBER OF WRITTEN DATA BYTES:15
POINTER TO PREVIOUS BLOCK
POINTER TO NEXT BLOCK
ENCRYPTED DATA
LOCATION:0
NUMBER OF DATA BYTES:4
NUMBER OF WRITTEN DATA BYTES:4
POINTER TO PREVIOUS BLOCK
NULL

FIG.2D

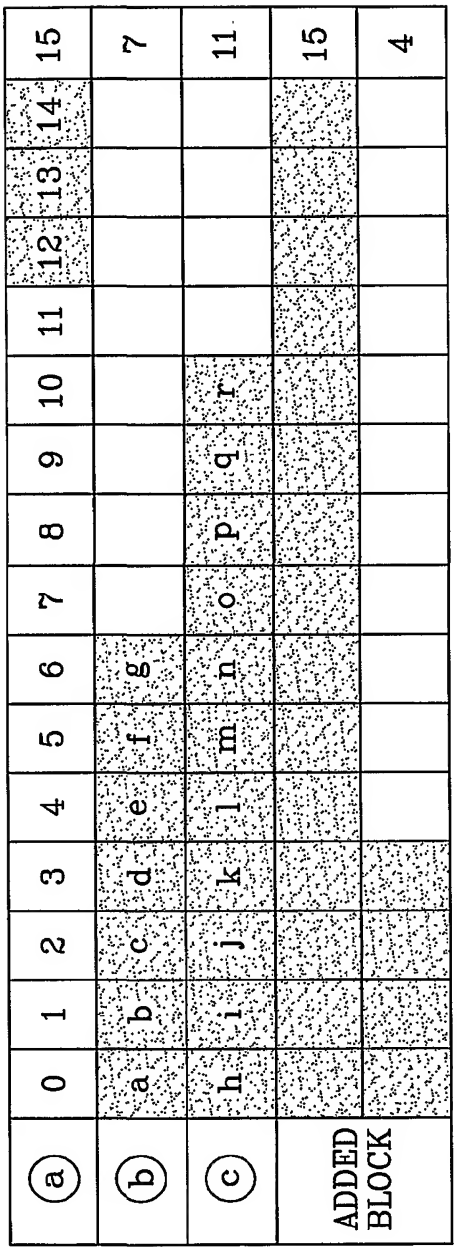

ⓐ
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
ⓑ
a b c d e f g
7
ⓒ
h i j k l m n o p q r
11
ADDED BLOCK
15
4

FIG.3A

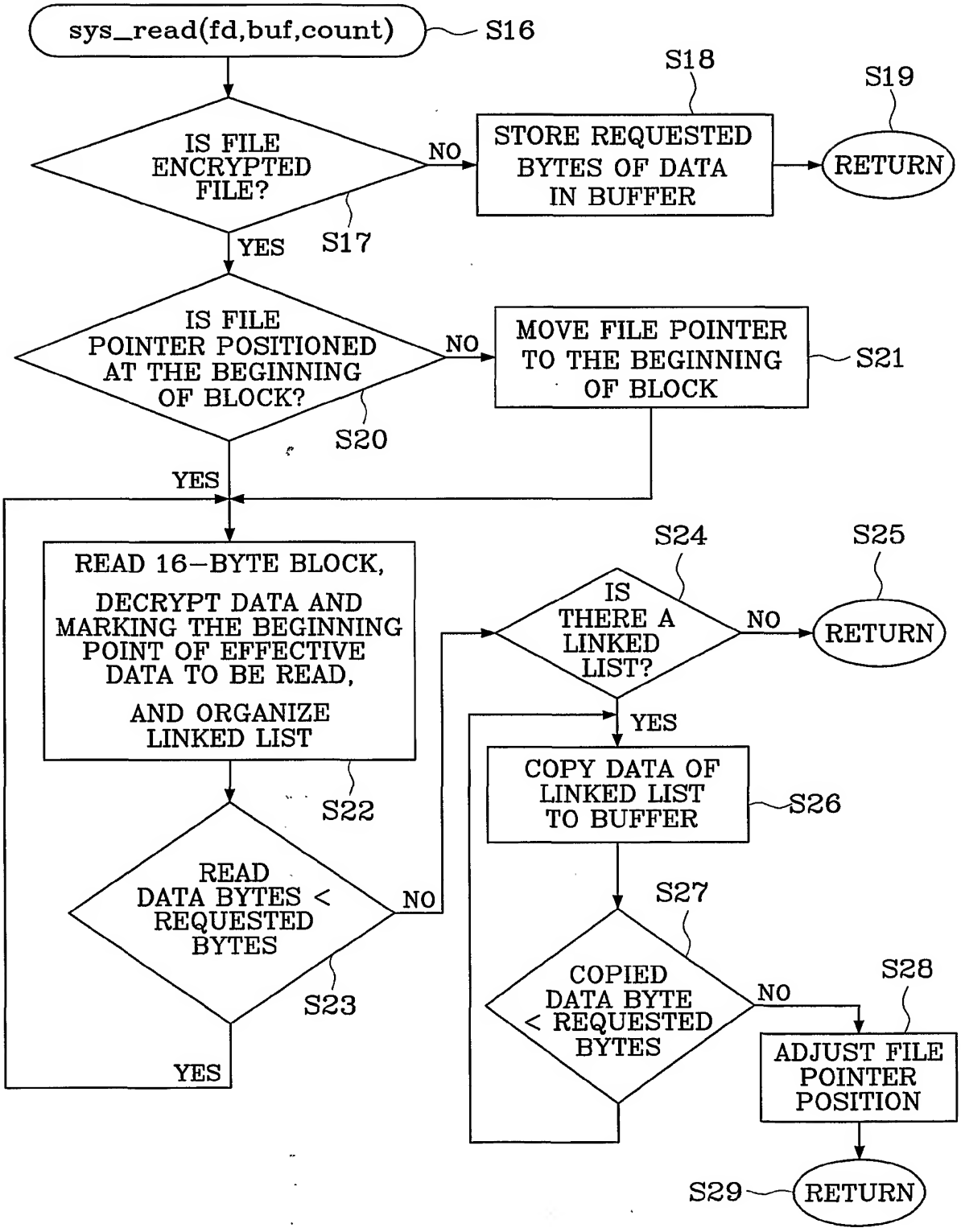

sys_read(fd,buf,count)
S16
IS FILE ENCRYPTED FILE?
NO
S18
STORE REQUESTED BYTES OF DATA IN BUFFER
S19
RETURN
YES
S17
IS FILE POINTER POSITIONED AT THE BEGINNING OF BLOCK?
NO
MOVE FILE POINTER TO THE BEGINNING OF BLOCK
S21
S20
YES
READ 16-BYTE BLOCK, DECRYPT DATA AND MARKING THE BEGINNING POINT OF EFFECTIVE DATA TO BE READ, AND ORGANIZE LINKED LIST
S22
READ DATA BYTES < REQUESTED BYTES
NO
S23
YES
S24
IS THERE A LINKED LIST?
NO
S25
RETURN
YES
COPY DATA OF LINKED LIST TO BUFFER
S26
S27
COPIED DATA BYTE < REQUESTED BYTES
NO
S28
ADJUST FILE POINTER POSITION
S29
RETURN

FIG.3E

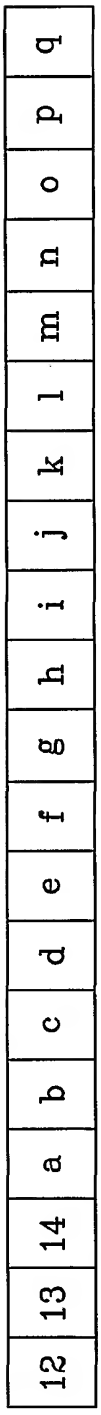

12
13
14
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q

FIG.3F

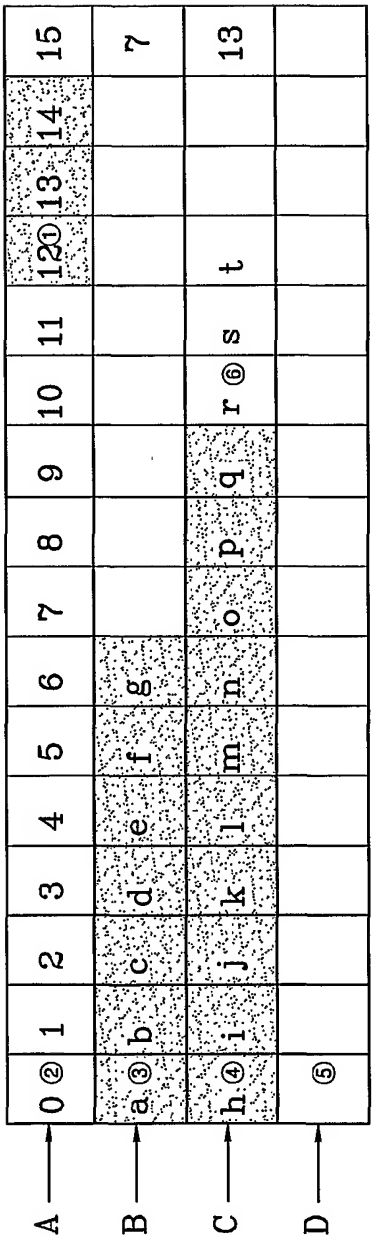

0 ② 1 2 3 4 5 6 7 8 9 10 11 ⑫ 13 14 15
a ③ b c d e f g
h ④ i j k l m n o P q r ⑥ s t 13 7
⑤
A
B
C
D

FIG.3B

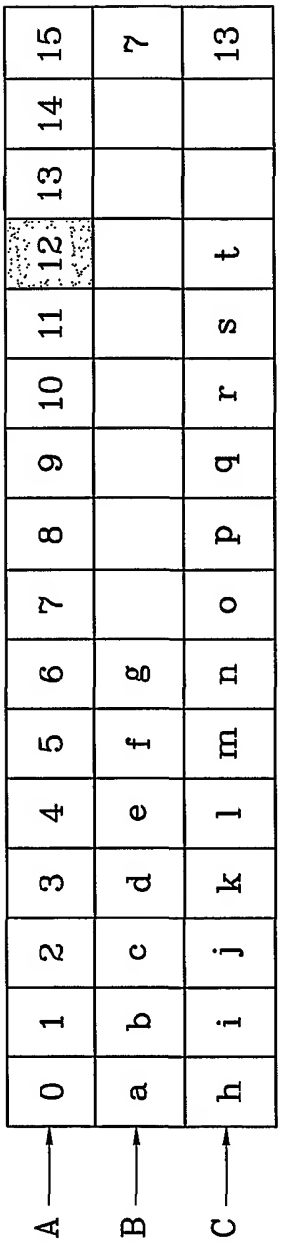

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
A
a b c d e f g
7
B
h i j k l m n o p q r s t
13
C

FIG.3C

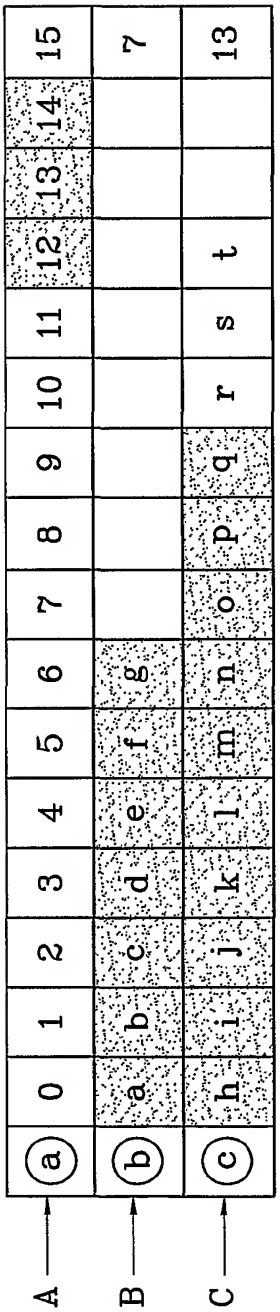

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
a
a b c d e f g
7
b
h i j k l m n o p q r s t
13
c
A
B
C

FIG.3D

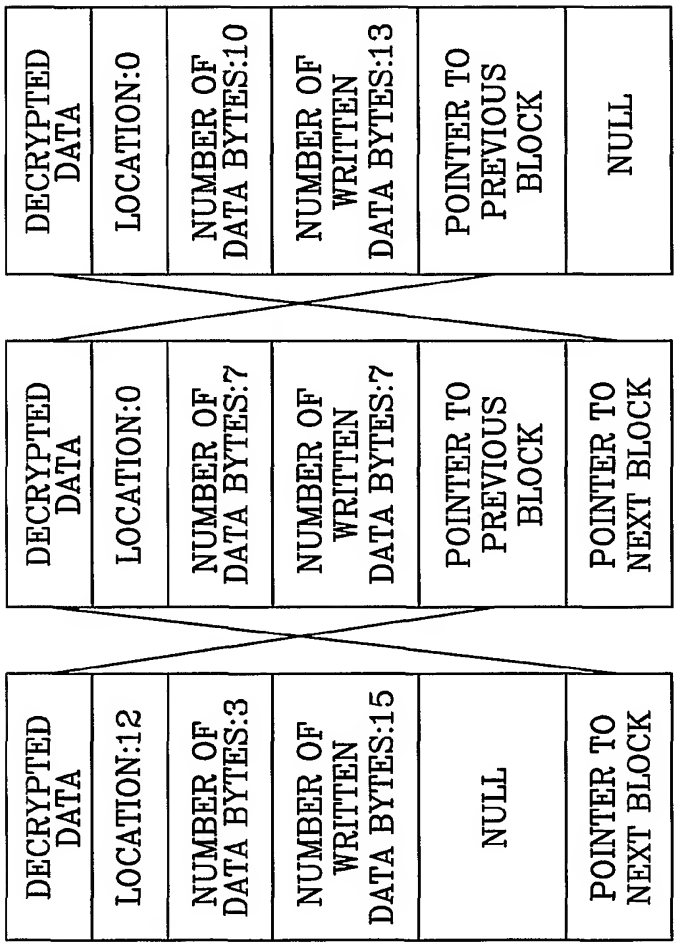

ⓐ
DECRYPTED DATA
LOCATION:12
NUMBER OF DATA BYTES:3
NUMBER OF WRITTEN DATA BYTES:15
NULL
POINTER TO NEXT BLOCK
ⓑ
DECRYPTED DATA
LOCATION:0
NUMBER OF DATA BYTES:7
NUMBER OF WRITTEN DATA BYTES:7
POINTER TO PREVIOUS BLOCK
POINTER TO NEXT BLOCK
ⓒ
DECRYPTED DATA
LOCATION:0
NUMBER OF DATA BYTES:10
NUMBER OF WRITTEN DATA BYTES:13
POINTER TO PREVIOUS BLOCK
NULL

FIG.4A

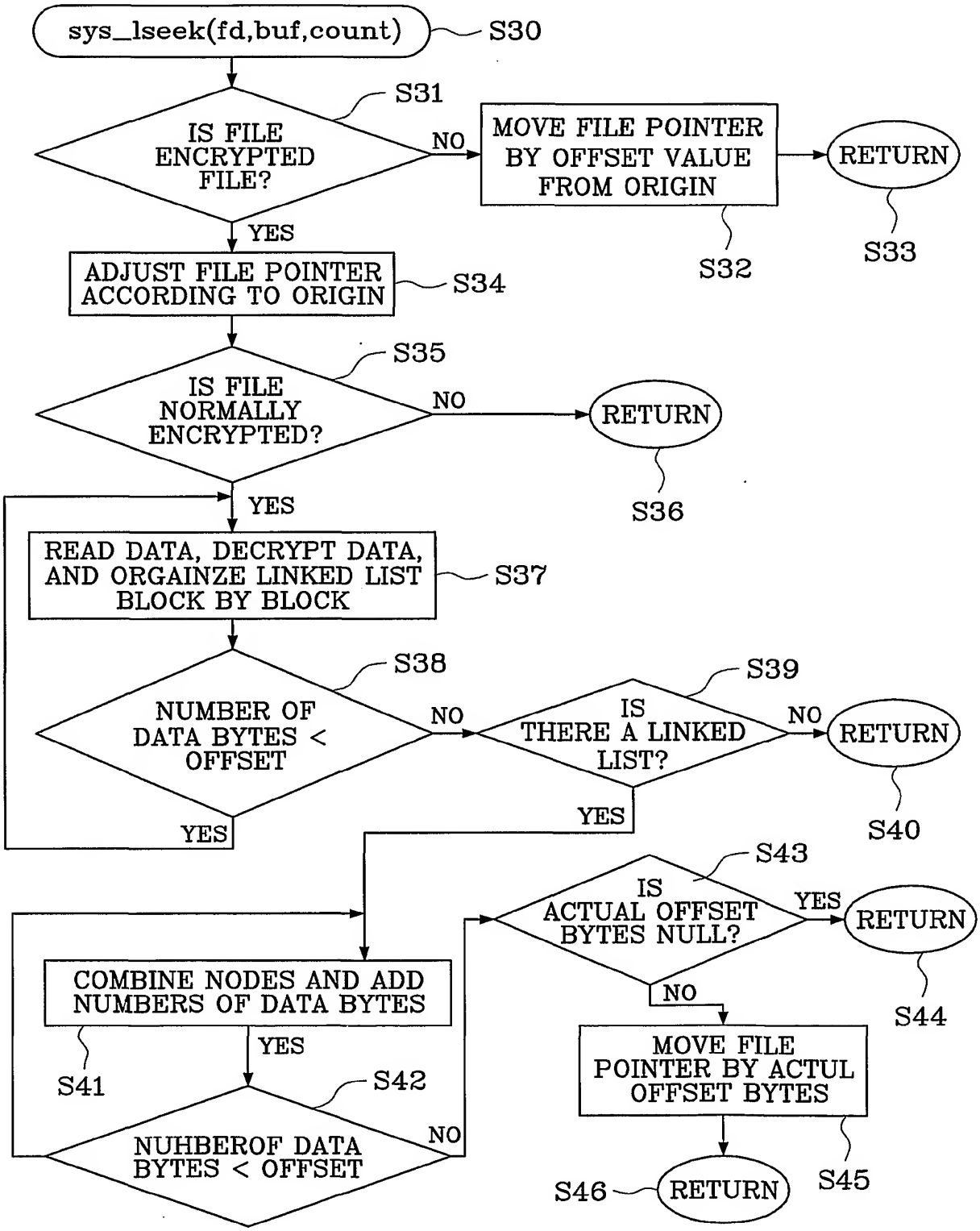

sys_lseek(fd,buf,count)
S30
S31
IS FILE ENCRYPTED FILE?
NO
MOVE FILE POINTER BY OFFSET VALUE FROM ORIGIN
RETURN
S32
S33
YES
ADJUST FILE POINTER ACCORDING TO ORIGIN
S34
S35
IS FILE NORMALLY ENCRYPTED?
NO
RETURN
S36
YES
READ DATA, DECRYPT DATA, AND ORGAINZE LINKED LIST BLOCK BY BLOCK
S37
S38
NUMBER OF DATA BYTES < OFFSET
NO
S39
IS THERE A LINKED LIST?
NO
RETURN
S40
YES
YES
S43
IS ACTUAL OFFSET BYTES NULL?
YES
RETURN
S44
NO
COMBINE NODES AND ADD NUMBERS OF DATA BYTES
S41
YES
S42
NUHBEROF DATA BYTES < OFFSET
NO
MOVE FILE POINTER BY ACTUL OFFSET BYTES
S45
S46
RETURN

FIG.4B

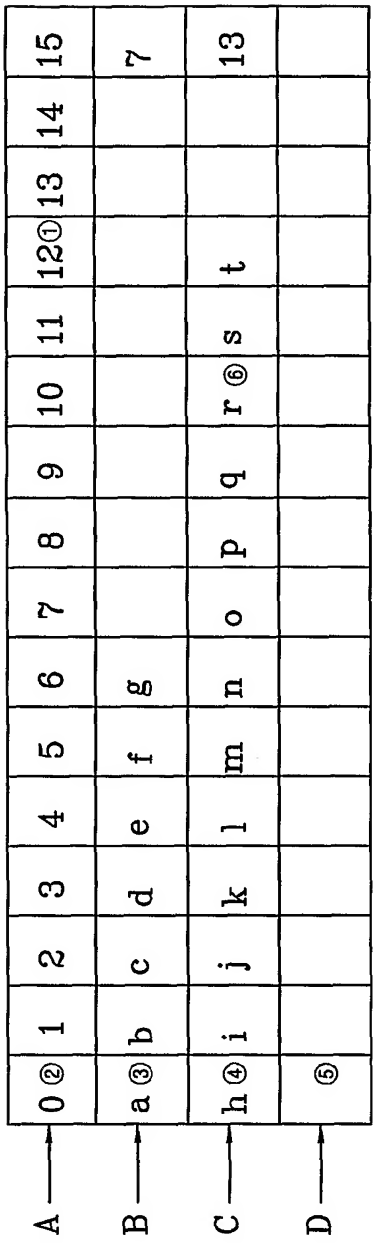

0 ② 1 2 3 4 5 6 7 8 9 10 11 12① 13 14 15
a ③ b c d e f g 7
h ④ i j k l m n o p q r ⑥ s t 13
⑤
A
B
C
D

FIG.4C

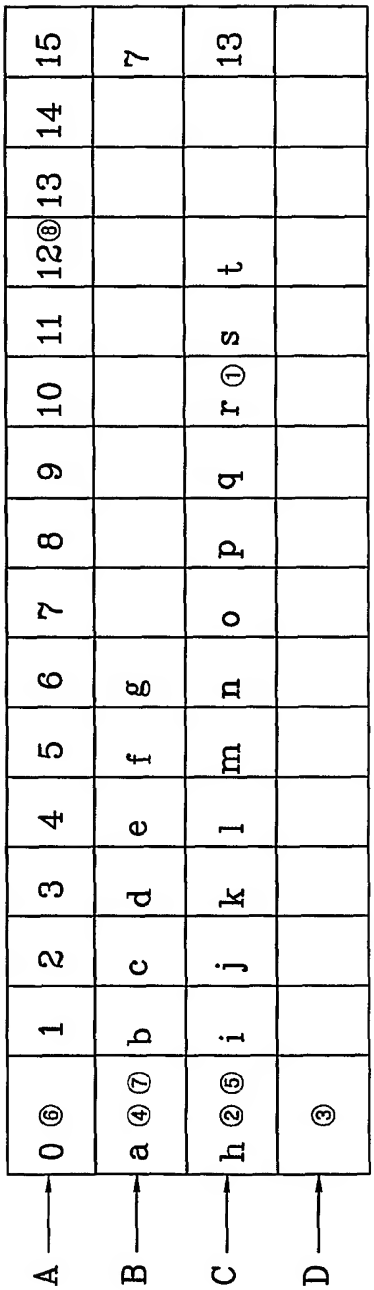
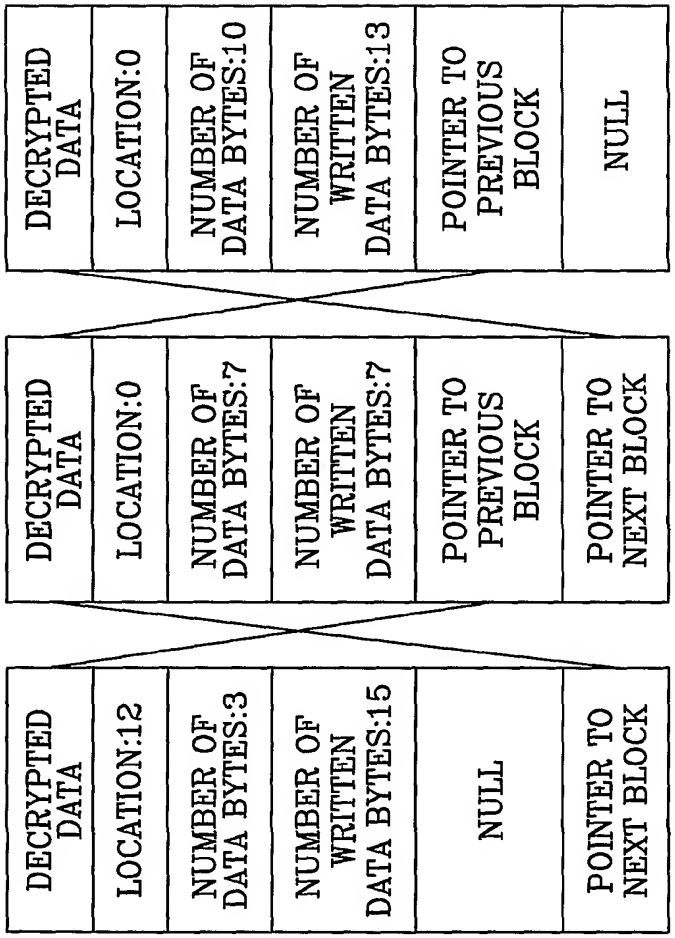

0 ⑥ 1 2 3 4 5 6 7 8 9 10 11 12⑧ 13 14 15
a ④ ⑦ b c d e f g 7
h ② ⑤ i j k l m n o p q r ① s t 13
③
A
B
C
D

FIG.4D


ⓐ
DECRYPTED DATA
LOCATION:12
NUMBER OF DATA BYTES:3
NUMBER OF WRITTEN DATA BYTES:15
NULL
POINTER TO NEXT BLOCK
ⓑ
DECRYPTED DATA
LOCATION:0
NUMBER OF DATA BYTES:7
NUMBER OF WRITTEN DATA BYTES:7
POINTER TO PREVIOUS BLOCK
POINTER TO NEXT BLOCK
ⓒ
DECRYPTED DATA
LOCATION:0
NUMBER OF DATA BYTES:10
NUMBER OF WRITTEN DATA BYTES:13
POINTER TO PREVIOUS BLOCK
NULL